```c
/*************************************************************************
 * Program: DoubleDim.c
 * Author: Sanjay Vyas
 *
 * Description
 *  This program shows how to create double dimension arrays and
 *  allocate memory for each type.
 *************************************************************************/

#include <stdio.h>
#include <stdlib.h>

#define R (3)
#define C (3)

int main()
{
    int aa[R][C];   // Array of array
    int *ap[R];     // Array of pointers
    int (*pa)[C];   // Pointer to array
    int **pp;       // Pointer to pointers

    int i;          // Loop variable for first dimension
    int j;          // Loop variable for second dimension

    int val;        // User input

    // 1. aa - Array of array required no allocation

    // 2. ap - Array of pointers need to allocate columns
    for (i = 0; i < R; i++)
    {
        ap[i] = (int *)malloc(C * sizeof(int));
        if (NULL == ap[i])
        {
            fprintf(stderr, "Sorry, not enough memory\n");
            return 1;
        }
    }

    // 3. pa - Pointer to array needs to allocate rows
    pa = (int (*)[C])malloc(R * sizeof(int[C]));
    if (NULL == pa)
    {
        fprintf(stderr, "Sorry, not enough memory\n");
        return 1;
    }

    // 4. pp - Pointer to pointer requires allocation for rows and columns
    pp = (int**)malloc(R * sizeof(int*));
    if (NULL == pp)
    {
        fprintf(stderr, "Sorry, not enough memory\n");
        return 1;
    }
    for (i = 0; i < R; i++)
    {
        pp[i] = (int *)malloc(C * sizeof(int));
        if (NULL == pp[i]) {
            fprintf(stderr, "Sorry, not enough memory\n");
            return 1;
        }
    }

    // Read values from the user and print all 4 double dimension
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
        {
            printf("Enter value for Element[%d][%d]: ", i, j);
            scanf("%d", &val);

            // Assign the same value to all 4 double dim
            aa[i][j] = ap[i][j] = pa[i][j] = pp[i][j] = val;
        }
    }

    // Print array of array
    printf("Printing array of array\n");
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
            printf("%d\t", aa[i][j]);
        printf("\n");
    }
    printf("\n");

    // Print array of pointers
    printf("Printing array of pointers\n");
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
            printf("%d\t", ap[i][j]);
        printf("\n");
    }
    printf("\n");

    // Print pointer of array
    printf("Printing pointer to array\n");
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
            printf("%d\t", pa[i][j]);
        printf("\n");
    }
    printf("\n");

    // Print pointer of pointer
    printf("Printing pointer to pointer\n");
    for (i = 0; i < R; i++)
    {
        for (j = 0; j < C; j++)
            printf("%d\t", pp[i][j]);
        printf("\n");
    }
    printf("\n");


    // Free up memory for ap, pa and pp
    // 1. aa does not require free as it required no malloc


    // 2. Free up for ap.. it requires a loop
    for (i = 0; i < R; i++)
        free(ap[i]);

    // 3. Free up for pa.. it requires a single free
    free(pa);

    // 4. Free up pp.. it requires a loop and then one free for top level
    for (i = 0; i < R; i++)
        free(pp[i]);

    free(pp);

    return 0;
}

```